# DBMS_METADATA und DBMS_METADATA_DIFF im Praxiseinsatz

**Philipp Loer**
**ORDIX AG**
**Paderborn**

**Schlüsselworte**

DBMS_METADATA, DBMS_METADATA_DIFF, SXML, AUDIT

**Abstract**

Ziel dieses Vortrags ist es aufzuzeigen, wie durch die Nutzung von DBMS_METADATA, DBMS_METADATA_DIFF und Auditing Datenbankmigrationsskripte inkl. DDL-Rollback Statements automatisiert generiert werden können.

Der Vortrag gliedert sich dabei in zwei Teile. Im ersten Teil wird die grundsätzliche Funktionsweise der Packages DBMS_METADATA und DBMS_METADATA_DIFF sowie Auditing vorgestellt. Im zweiten Teil des Vortrags erfolgt ein Praxisbericht und eine Live-Demo, wie mit Hilfe der vorgestellten Packages Rollback-Skripte für DDL-Statements generiert werden können.

Durch den kombinierten Einsatz der Technologien DBMS_METADATA, DBMS_MEDATA_DIFF und AUDIT ist es möglich sämtliche Schemaveränderungen durch DDL- und DML-Statements zu protokollieren. Im Rahmen der Softwareentwicklung ermöglicht die Kombination dieser Technologien die automatisierte Erstellung von Datenbankmigrationsskripten von einer Datenbankversion auf die nächste und darüber hinaus auch die automatisierte Erstellung von DDL-Rollback-Skripten.

In einem unserer Kundenprojekte verwenden wir das Package DBMS_METADATA_DIFF, um die Veränderungen eines Schemas über die Laufzeit des Projektes nachzuverfolgen. Hierzu wurde in einem Master-Schema ein DDL-Trigger geschrieben. Dieser Trigger speichert vor jeder Veränderung eines Objektes dessen aktuellen Zustand in Form eines SXMLs ab. Durch den Vergleich zweier SXMLs verschiedener Objektversionen kann ein ALTER DDL generiert werden, dass es ermöglicht, ein Objekt aus der Softwareversion X auf die Softwareversion Y zu migrieren. Gleichzeitig wird ein weiteres ALTER-DDL-Statement (Rollback) generiert, um genau diese Änderung wieder rückgängig machen zu können.

Bei einer Schemamigration ist es allerdings nicht ausreichend, alleine Veränderungen am Schema durchzuführen. Es müssen auch sämtliche DML-Statements im beschriebenen Master Schema gespeichert und auf den Zielsystemen ausgeführt werden. Da es zu aufwändig wäre für jede Tabelle des Master-Schemas einen DML-Trigger zu schreiben, empfiehlt sich hier die Verwendung von Auditing. Durch dessen Einsatz werden auch die Datenbankmigrationen des Master-Schemas protokolliert und können so in die generierten Migrationsskripte integriert werden. Während des Vortrags wird dieses Vorgehen in einer Live-Demo veranschaulicht.

Im Folgenden werden nun die im Einzelnen verwendeten Technologien kurz vorgestellt.

**SXML: Ein SQL in XML-Form**

Bereits seit Oracle 9i ist das Package DBMS_METADATA verfügbar. Dieses ermöglicht es, die gesamten Eigenschaften eines Objektes entweder als DDL-Statement oder als XML aus dem Data Dictionary auszulesen. Ein mit DBMS_METADATA generiertes XML-Abbild beinhaltet die gesamte strukturelle und logische Komplexität des Data Dictionary. Zusätzlich beinhaltet das mit DBMS_METADATA generierte XML spezifische Werte des Data Dictionary der Instanz.

Mit Oracle 11g Release 2 wurde dieses Package erweitert. Zusätzlich ist es ab dieser Version möglich, neben SQL und XML auch die Repräsentation eines Objektes im SXML-Format zu generieren. Ein SXML ist eine „human-readable"-Version des bereits länger verfügbaren XML. Es enthält keinerlei instanzspezifische Werte mehr, da es im Gegensatz zum XML kein Abbild des Data Dictionary, sondern ein Abbild des DDL-Statements eines Objektes in XML-Struktur ist. Die XML-Tags entsprechen den gewohnten Bezeichnungen in der Oracle-Datenbank wie z.B. Schema, Column oder Datatype (siehe Abbildung 1).

```
SELECT dbms_metadata.get_sxml('TABLE', EMP) from dual;
<TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">
  <SCHEMA>SCOTT</SCHEMA>
  <NAME>EMP</NAME>
  <RELATIONAL_TABLE>
    <COL_LIST>
    <COL_LIST_ITEM>
      <NAME>EMPNO</NAME>
      <DATATYPE>NUMBER</DATATYPE>
      <PRECISION>4</PRECISION>
      <SCALE>0</SCALE>
      <NOT_NULL/>
    </COL_LIST_ITEM>
…
```

*Abbildung 1: SXML*

Ein SXML kann daher auch durch Fremdsoftware generiert oder angepasst werden. Zudem ist es einfach möglich, zwei verschiede SXMLs zu vergleichen und die Unterscheide zu ermitteln.

**DBMS_METADATA_DIFF**

Der Vergleich von zwei SXMLs kann mit dem ebenfalls mit Oracle 11g Release 2 eingeführten Package DBMS_METADTA_DIFF vorgenommen werden. Ergebnis dieses Vergleichs ist ein um die Unterschiede angereichertes, wie in Abbildung 2 dargestelltes SXML (ein sogenanntes Compare SXML).

```
CREATE TABLE ordix1 (a number, b char(15));
CREATE TABLE ordix2 (b char(20));

<TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">
<SCHEMA>ORDIX</SCHEMA>
<NAME value1="ORDIX1">ORDIX2</NAME>
<RELATIONAL_TABLE>
  <COL_LIST>
    <COL_LIST_ITEM src="1">
      <NAME>A</NAME>
      <DATATYPE>NUMBER</DATATYPE>
    </COL_LIST_ITEM>
    <COL_LIST_ITEM>
```

```
        <NAME>B</NAME>
        <DATATYPE>VARCHAR2</DATATYPE>
        <LENGTH value1="10">20</LENGTH>
      </COL_LIST_ITEM>
   </COL_LIST>
</RELATIONAL_TABLE>
</TABLE>
```

*Abbildung 2: Compare SXML*

In diesem Beispiel ist in der ersten Tabelle die Spalte A vorhanden, in der zweiten nicht. Die Spalte B hat zwar den gleichen Datentyp, unterscheidet sich jedoch in der Länge. Eine Umwandlung des SXML in ein DML liefert folgendes Ergebnis:

```
ALTER TABLE ORDIX.ORDIX1 DROP (A);
ALTER TABLE ORDIX.ORDIX1 MODIFY (B CHAR(20));
ALTER TABLE ORDIX.ORDIX1 RENAME TO ORDIX2;
```

Eine Generierung dieser ALTER-DDLs mit DBMS_METADATA_DIFF ist genauso einfach wie gefährlich. Das erste Statement in diesem Beispiel löscht eine Spalte inklusive der dort enthaltenen Daten. Das dritte Statement versucht die Tabelle ORDIX1 in ORDIX2 umzubenennen. Dies führt, da sich beide Tabellen im gleichen Schema befinden, unweigerlich zu einem Fehler.

Es ist daher immens wichtig, dass die generierten Statements vor ihrer Ausführung sorgfältig überprüft werden. DBMS_METADATA_DIFF nimmt keine Rücksicht auf verlorene Daten oder Performance-Gesichtspunkte.

**Customizing**

Dem Package DBMS_METADATA können gewisse Regeln für die Generierung mitgegeben werden. Diese Regeln ermöglichen es, sowohl das CREATE -DDL-Statement eines Objektes als auch ein ALTER-DDL-Statement individuell anzupassen. Hierzu ist es notwendig, die einzelnen Schritte der Generierung in PL/SQL nachzubilden. Im Folgenden legen wir den Fokus auf die Anpassung eines ALTER DDL. Die Ausführungen gelten jedoch analog für die Anpassung eines DDL-Statements zur Erzeugung eines Objektes.

Zunächst müssen aus den beiden zu vergleichenden Objekten jeweils SXMLs generiert werden. Mit Hilfe der Funktion DBMS_METADATA_DIFF.COMPARE_SXML wird nun aus den beiden SXMLs ein Compare SXML generiert. Anschließend wird das Compare SXML zunächst in ein ALTER SXML und schließlich in ein oder mehrere ALTER DDLs umgewandelt (siehe Abbildung 3).


*Abbildung 3: Der Weg vom Vergleich zum Alter DDL*

Über die folgenden Prozeduren ist es möglich ein SXML individuell anzupassen:
Mit Hilfe der Prozedur SET_PARSE_ITEM ist es möglich, dass nicht per DEFAULT generierte Objekteigenschaften ebenfalls erzeugt werden. Dies sind u.a. abhängige Objekte wie Indizes, Trigger oder der Tablespace des Objektes.

Durch die Prozedur SET_REMAP_PARAM ist es möglich, die Bezeichner einiger Objekteigenschaften, wie z.B. den Tablespace- oder Schemanamen, individuell anzupassen. Weitere Eigenschaften der generierten DDL-Statements können über die Prozedur SET_TRANSFORM_PARAM bestimmt werden. Hier kann u.a. die STORAGE-CLAUSE, die Generierung nicht referentieller Constraints oder auch eines Packages Body abgeschaltet werden.

```
DBMS_METADATA.SET_PARSE_ITEM(openw_handle, 'TABLESPACE');
DBMS_METADATA.SET_REMAP_PARAM(openw_handle, 'ORDIX', '&schema');
DBMS_METADATA.SET_TRANSFORM_PARAM(openw_handle, 'STORAGE', 'FALSE');
```

*Abbildung 4: Customizing*

Das hier gezeigte Vorgehen kann helfen, einige der Probleme bei der Generierung der ALTER DDLs zu verhindern. Bestimmte Problemkonstellationen (wie das Umbenennen einer Spalte) führen jedoch unweigerlich zu falschen Ergebnissen. Wird eine Spalte umbenannt, so kann DBMS_METDATA_DIFF dies nicht erkennen und generiert ein ALTER TABLE DROP COLUMN und ein Statement vom Typ ALTER TABLE ADD COLUMN. Diese Veränderung gleicht zwar die beiden Tabellen einander an, die Daten der umbenannten Spalte sind jedoch verloren.

**Auditing**

Im Datenbankumfeld versteht man unter Auditieren das Aufzeichnen von Benutzeraktivitäten. Auditing ist seit Oracle 6 verfügbar, seit 11g ist es standardmäßig für sicherheitsrelevante Befehle aktiviert.

Der Initialisierungsparameter AUDIT_TRAIL ist standardmäßig auf den Wert „db“ gesetzt. D.h. die Auditing-Daten werden in der Datenbank gespeichert, genauer gesagt in der Tabelle AUD$. Andere mögliche Parameter sind „bs“ (Speicherung im Betriebssystem) oder „xml“ (Speicherung im Betriebssystem im XML-Format). Um nicht nur die Art des SQL-Statements, den ausführenden User etc. sondern auch das Statement selbst zu sichern, muss das Auditing im EXTENDED-Modus betrieben werden. Daher ist der Parameter AUDIT_TRAIL auf den Wert „db, extended“ anzupassen. Ebenfalls möglich ist der Wert „xml, extended“, jedoch ist hier die Speicherung in der Datenbank für den hier beschriebenen Anwendungsfall sinnvoller.

Nachdem der Initialisierungsparameter AUDIT_TRAIL gesetzt und aktiviert ist, wird mit dem Befehl AUDIT festgelegt, welche Aktionen konkret auditiert werden sollen. Durch den folgenden Befehl kann das Auditing auf einer Tabelle aktiviert werden:

```
AUDIT ALL ON tabelle1;
```

Um das Auditing auch für alle Tabellen eines Schemas zu aktivieren kann entweder das Auditing für alle User mit Rechten in diesem Schema aktiviert werden. Alternativ ist es möglich, das Auditing für bereits bestehende Objekte mit dem oberen Befehl zu aktivieren und für zukünftige Objekte durch eine Anpassung der Default-Audit-Parameter eine Auditierung vorzuschreiben (siehe Abbildung 5).

```
AUDIT INSERT TABLE, UPDATE TABLE, DELETE TABLE BY master_schema;
AUDIT ALTER, GRANT, INSERT, UPDATE, DELETE ON DEFAULT;
```

*Abbildung 5: User Audit / Default Audit*

Die Audit-Daten können direkt aus der Tabelle AUD$ bzw. vom Betriebssystem ermittelt werden. Es gibt aber einige Views, die das Lesen der Daten vereinfachen.

Im Folgenden einige Beispiele:

```
DBA_AUDIT_TRAIL  -- alle Einträge in aud$
DBA_OBJ_AUDIT_OPTS  -- alle Auditing Optionen, die für Objekte aktiviert
sind
```

**Kontaktadresse:**

Philipp Loer
ORDIX AG
Westernmauer 12-16
D-33098 Paderborn

Telefon: +49 (0) 1063-0
Fax: +49 (0) 180-1 67349 0
E-Mail info@ordix.de
Internet: www.ordix.de